LSAI Project Feature: Flash-Attention

Ahmad Khan
ETH Zurich
ahkhan@ethz.ch

May 2025

1 Feature Description

In this feature, we enable Flash-Attention [Dao+22] for training our llama-like [Tou+23] model. FlashAttention speeds up the computation of exact attention by reducing memory read/write operations. This is achieved by leveraging tiling and GPU-friendly computation. By replacing the standard attention mechanism with FlashAttention, we aim to improve throughput and training speed without compromising model accuracy.

2 Experiment Design

To evaluate the impact of FlashAttention, we conduct training experiments on the Swiss Alps supercomputer cluster [Swi24]. The model is trained with a sequence of length 4096 using a batch size of 1, a learning rate of 5e-5, and a warm-up period of 100 steps, for a total of 1000 training steps. We record metrics including throughput, TFLOPs, and loss at regular intervals.

To compare the performance of different attention algorithms, we perform 5 different experiments with 5 different backends:

1. Baseline: unchanged starter code
2. Efficient attention
3. CUDNN attention
4. FlashAttention2
5. FlashAttention3

For the baseline, since we use `scaled_dot_product_attention`[1], PyTorch automatically selects one of the optimized attention backends by default. We

[1] `https://pytorch.org/docs/stable/generated/torch.nn.functional.scaled_dot_product_attention.html`

confirm this behavior in a separate experiment by explicitly specifying the basic `Math` backend, which results in an out-of-memory error. Based on our results in section 3, we believe that the baseline is automatically selecting the FlashAttention2 backend.
For Efficient, CUDNN, and FlashAttention2 we use the pytorch implementation[2]. For FlashAttention3 we use the implementation provided by the authors of FlashAttention[3].

All the code to reproduce our results can be found in our codebase[4].

3 Results

In this section, we compare all five attention backends across several dimensions.

3.1 Training Loss

Here we look at the training loss across the different backends. The results are show in fig. 1. We can see that all the backends achieve similar losses. This is to be expected since FlashAttention is an exact attention computation, and not an approximation. The only line that diverges a little is FlashAttention3 at the beginning. This is likely because it uses an entirely different non-pytorch implementation, and thus might have some internal differences. However, in the end the loss values are very comparable.

Figure 1: Training loss for different attention backends

[2]https://docs.pytorch.org/docs/stable/generated/torch.nn.attention.SDPBackend.html
[3]https://github.com/Dao-AILab/flash-attention
[4]https://github.com/mak2508/lsai-proj

3.2 Tokens per second

Next we look at the tokens per second in fig. 2. Tokens per second measures the throughput of the model during training, indicating how many input tokens are processed per second—a higher value reflects faster training performance. This is relevant since it is an indicator of how quickly the backend is computing attention. In our results, we see that FlashAttention3 is the most performant, which is within our expectations. CUDNN Attention seems to be also more optimized than FlashAttention2. We can further see that the baseline is likely using FlashAttention2 since the lines match quite closely.

Figure 2: Tokens per second for different attention backends

3.3 Model FLOPs Utilization

Model FLOPs Utilization (MFU) measures the percentage of a GPU's theoretical peak floating-point operations per second (FLOPs) that are actually used by the model during training. It is a key efficiency metric that reflects how well the model implementation leverages hardware capabilities, with higher MFU indicating better performance utilization. This is presented in fig. 3. Here, we see again that FlashAttention3 performs the best, which is expected since it is optimized for the Grace Hopper GPUs we have access to in Alps.

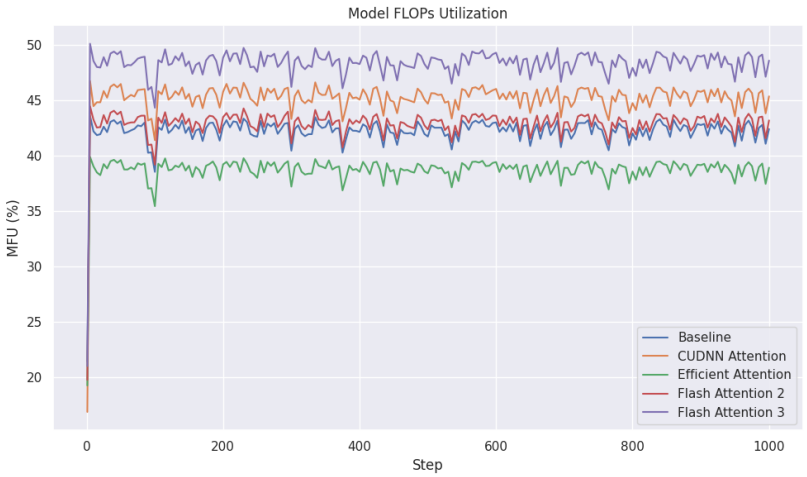

Figure 3: Model Flop Utilization (MFU) over training

3.4 Training Time

Finally, we look at the training time for each of the backends. We see in fig. 4 again, inline with the previous two results, that FlashAttention3 performs the best.

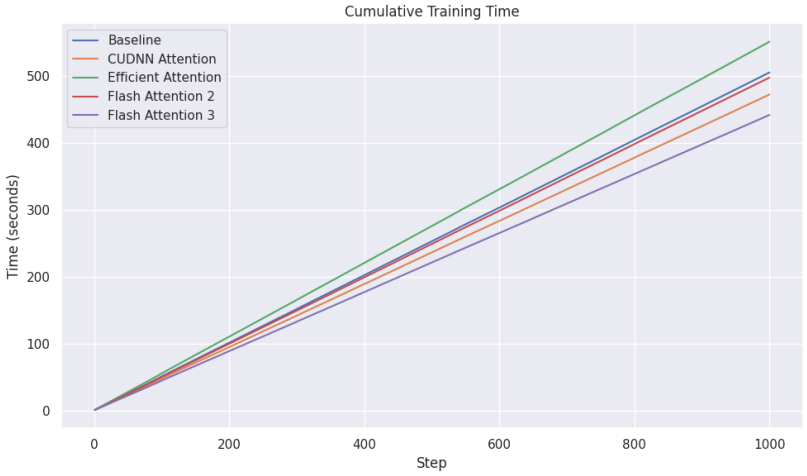

Figure 4: Training time across backends